Context Switching and IPC Performance Comparison between uClinux and Linux on the ARM9 based Processor

Hyok-Sung Choi, Hee-Chul Yun
{hyok.choi, heechul.yun}@samsung.com
Software Platform Lab, Digital Media R&D Center, Samsung Electronics

Abstract

uClinux is a derivation of Linux kernel intended for MMU-less processors. It provides a single shared address space for all processes while the Linux kernel provides a separate virtual address space for each process using hardware MMU (memory management unit). In this paper, we implemented Linux and uClinux kernels on the same ARM9 platform and compared the performance. The ARM9 processor features virtually indexed caches and a TLB without address space tag. Therefore Linux should flush entire cache and TLB on each context switch which is very costly. uClinux, however, contents of caches and a TLB are valid even after context- switch because the same address space is shared among all processes. We observed an order of magnitude reduction of the context switching overheads on uClinux. As a result, IPC (Inter Process Communication) performance is also better on uClinux.

Keyword
Embedded OS, Linux, uClinux, benchmark, MMU, cache, performance, context switching, IPC, ARM

1. INTRODUCTION

Context switching is the series of procedures to switch the control of CPU from current process to a certain process. While the context switching, the operating system saves the context of current process and restores the context of the next process which is decided by some certain scheduling policy.

The context switching time of an operating system which supports the virtual address space, vary to the algorithms of cache of the architecture. Cache architecture can be split by the addressing scheme as virtual cache and physical cache.[1] The virtual cache stores the location of a certain context with virtual address and the physical cache stores with physical address. The access time for the virtual cache could be better for the simplicity of comparison with the context of cache which needs no address conversion. However, for context switching, the whole context of the virtual cache must be flushed and invalidated because of the meaningless after the switching. For example, ARM9 architecture has virtual address based cache and after the context switching the whole memory access makes cache miss at every each first access time which makes heavy load.

The uClinux [2] is designed for MMU-less architectures at first. On uClinux, one singular address space is shared by the whole processes which had its own virtual address space while on Linux. It makes difficult to support memory protection and vast address space but the load for context switching and communication can be much smaller while supporting the whole compatibility with Linux API except for fork() which is replaced by vfork() in uClinux.

In this paper, we analyzed the virtual addressing cache architecture of the ARM9 which is mostly used platform for embedded systems, and compared the expense of time of context switching for uClinux and Linux on the same platform.

The cache and TLB architecture is described at the chapter 2 and the implementation issues for uClinux and Linux is described at the chapter 3. The benchmark environment and the program are described at the chapter 4 and the benchmark result is described at the chapter 5. The previous works summarized at the chapter 6.

2. STRUCTURAL ANALYSIS

The structure of the cache and TLB (Translation Look-aside Buffer) of the MMU based ARM processor is as Fig.1. [8]

ARM architecture is designed as the harvard architecture. CPU outputs the virtual address(VA) and if the matching data is in the I(instruction) or D(data) cache return the context by cache-hit. If it is not in the cache, the TLB is used for matching the VA with the physical address(PA) and the cache-line is filled from the memory.

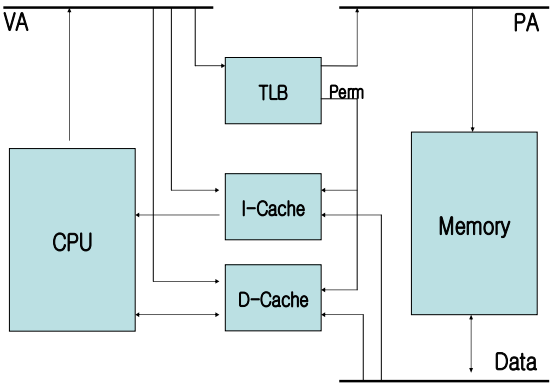

Fig. 1. The Cache and TLB architecture diagram of ARM processor.

Fig.2 shows the detailed cache structure of ARM9 processor [6][7]. The “Index” part from the 32bits virtual address which is made by CPU used for indexing of the “Tag” table and if the indexed entry’s tag information matches the tag information from the virtual address, the data from cache-line is accessed. In the Fig.2, the cache index and the tag itself are based on the virtual address. Thus, while process context switching time, the whole current cache context and TLB is invalidated if it is set by the WT(Write-Through) cache policy, even need to be flushed into the real memory if WB(Write-Back) cache policy is used.

For flushing the cache, about 1k ~ 18k CPU cycle is needed depend on the cache size and the side operations which is needed to fill-up the cache-line and TLB takes up to about 54k CPU cycle. For 200MHz ARM9 processor, it is about 270μs time, [3] and is a heavy burden for many real-time applications which needs under several tens of μs response delay.

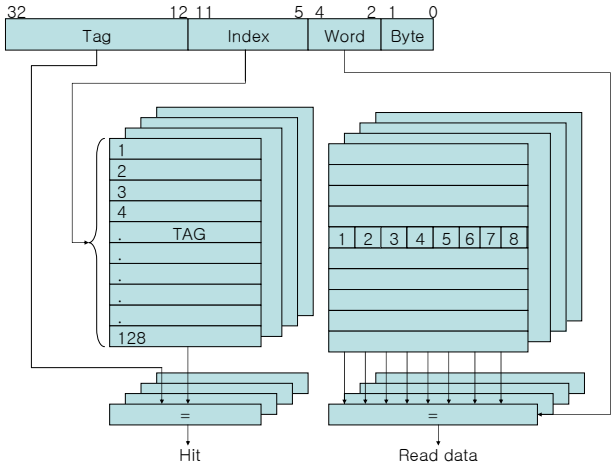

Fig. 2. The detailed cache architecture diagram based on virtual address. (ARM926EJ-S)

3. IMPLEMENTATION

uClinux is the modified Linux kernel for architectures which has a MPU(Memory Protection Unit) only or even no hardware memory management unit. From 2.0.x to 2.4.x, it is independently designed and developed from Linux kernel. However, from Linux 2.6.0 beta test versions, from the m68knommu architecture, it is merged into the mainstream Linux kernel and separation from conventional Linux kernel and uClinux is not valid. Although uClinux supports the singular addressing space for kernel and application processes, except for fork() and memory remapping only, the whole Linux API is fully compatible and used for many of real-world embedded system development for architectures which has no MMU or even it has, if the single space addressing has advantages.

uClinux is available for architecture which has MPU or even MMU. The ARM MMU supports both page-mapping for 1KB, 4KB, 64KB size and section-mapping for 1MB size. Linux uses 4KB paging-mapping for memory management. For uClinux, 1MB section-mapping can be used for simple virtual to physical mapped single address space. To say again, D and I-cache operation and mapping core of MMU still work but paging mapping for uClinux.

In this paper, ARM926EJ-S core based Samsung S3C24A0 processor reference platform is used for Linux and uClinux 2.6.7. [4]

4. EXPERIMENTAL SETUP

4.1 The benchmark programs

lmbench[5] is the well-known benchmark program for performance testing over UNIX related operating systems. In this paper, “lat_ctx,” “lat_fifo” and “bw_fifo” is used with some modifications.

The “lat_ctx” is for measuring the requirement time for context switching. Creating “N” processes and series of “N” pipes, it constructs “pipe-ring” which links all the processes. Each process accesses its own “k” KB independent memory and “token” is passed through the next pipe to the neighbor processes, which makes a series of synchronized context switching and measure the cycle delay time.

The “lat_fifo” is for measuring the requirement time for send and receive a token between 2 processes.

The “bw_pipe” is for measuring the bandwidth of “pipe” to send and receive through it.

To be the exactly same code, while the benchmark, we made modifications on fork and pipe to be vfork and name-pipe (FIFO).

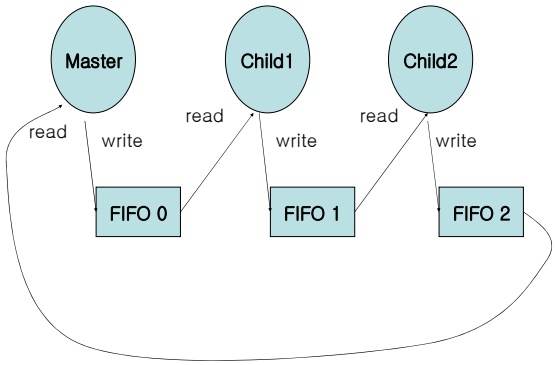

Fig. 3. The FIFO structure of the modified “lat_ctx”

Fig. 3 shows the modified FIFO structure for the “lat_ctx.” The parent processes creates FIFO which is sorted with the process numbers(PID). When the child processes are created by the vfork, the child processes open the neighbor FIFOs to be ready for the communication. Each of the all the child processes are blocked when it try to read the “n-1” FIFO and to be “sleep” state waiting for the write of the FIFO. If all the child processes are ready for read the FIFOs, the parent process writes a token to the first FIFO. It makes “wake” the first child process and the context switching is occurred, and the process writes the token to the next child process, and so on. This procedure makes the chain reaction to be context switched in series. If the last child process get the

CPU control and write to the last FIFO, the blocked parent process who was waiting for the input of the last FIFO is awaken and completes one cycle of the “pipe-ring.”

5. EVALUATION

5.1 The benchmark system

The S3C24A0 process which is used for the benchmark test is based on the ARM926EJ-S core and has 16KB D-Cache and 16KB I-Cache separately. The TLB has the capacity to store 64 entries simultaneously, and the I and D entries are not separated and the all of the caches and TLB is based on the virtual address.

The benchmark test is done on the same S3C24A0 platform and the same release version of kernel (2.6.7) for uClinux and Linux. And the benchmark program which is described at the Chapter 4 is used.

5.2 The benchmark result

The results of “lat_fifo” and “bw_pipe” are shown in the Table 1, which reflects the delayed time and the bandwidths of the FIFO.

The result shows that uClinux has 5 times and 2 times better performance than Linux on the delayed time and the bandwidths. This is from the cache operation of the Linux kernel which invalidates and flushes the whole caches for the context switching. In other hand, uClinux shares the one address space for all the processes include the kernel itself, and get the benefit of cache efficiency.

In other words, the difference of the IPC (Inter-Process Communication) performance can make a big performance improvement on the uClinux system applications which needs frequent processes communication.

TABLE 1
THE RESULTS OF THE IPC PERFORMANCE OF LINUX AND UCLINUX

	Linux	uClinux	Ratio
lat_fifo(μs)	160.64	31.74	5.06
bw_pipe(MB/s)	12.58	25.55	2.03

lat_fifo measures the delayed time of FIFO, smaller number is better.
bw_pipe measures the bandwidth of FIFO, bigger number is better.

The context switching delayed time is showed in the Fig. 4. The context switching time of uClinux and Linux varies from its own data access size (0KB, 1KB, 16KB) and the number of processes. When the access data size is 0KB, uClinux switches 4.5 times faster for 16 processes and 9.7 times for 2 processes.

For all cases of Linux settings, the context switching time is almost flat independently with the increase of number of processes. It shows the almost fixed cache miss burden which comes from the invalidation of the whole caches while the context switching. uClinux shows much smaller delayed time for context switching while the number of processes are small and increase depend on the number of processes. This comes from the decrease of the cache hit ratio of I-cache and D-cache, when the number of processes is increased. Especially when the process own data access size grows up to 16KB, the context switching time for both of uClinux and Linux almost same which comes from the hardware cache size of S3C24A0 which has 16KB I and D caches and the benefit from cache set off.

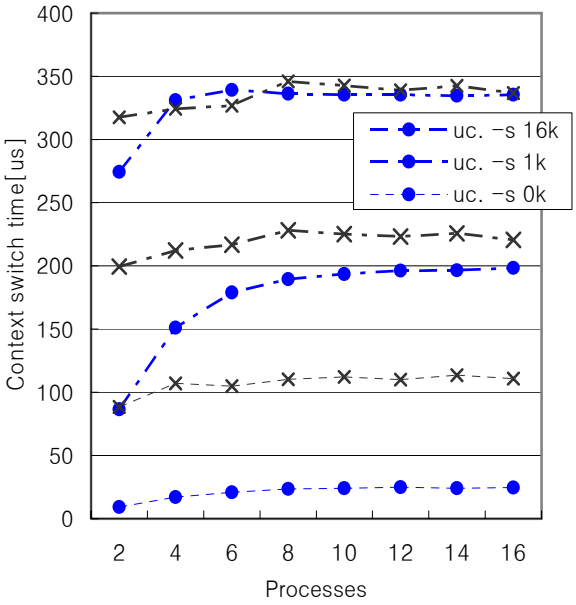

Fig. 4. The context switching time measured by “lat_ctx.” is shown. Each process has its own 0KB, 1KB or 16KB data access contents and the number of processes varies. The dot over the line “x” represents the value of Linux and “o” for the value of uClinux.

The magnified graph for own access data size 0KB is shown at Fig. 5. For Linux, all the caches are invalidated and flushed whenever switching the context and the cache-line is filled from the beginning. Thus, almost fixed cache-miss time load is needed irrelative to the number of processes.

In uClinux case, the possibility of cache hit occurrence is much stronger. However the delayed time increase while the number of process increase which makes increase the possibility of cache-miss because of the limited hardware cache capacity.

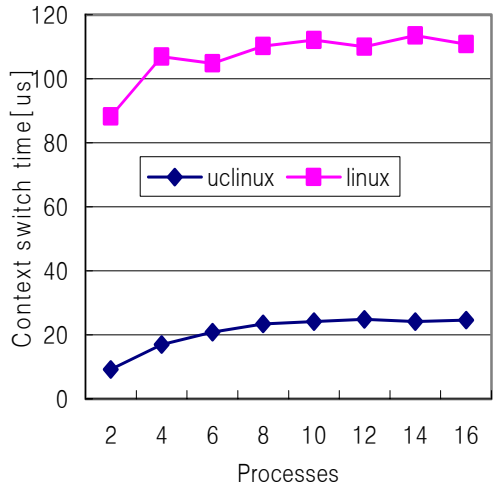

Fig. 5. The context switching time measured by “lat_ctx.” Each process has no its own data access storages. (size = 0KB)

The total gage R&R study on the benchmark system shows 0.25% contribution of VarComp and 5.03%SV.

6. RELATED WORK

Although we are not aware of any previous systematic study of the uClinux and Linux performance, many of extensions to operating systems performance improvement was introduced include real-time, sharing domain and overall performance.

In particular, many different real-time scheduling algorithms introduced and have been implemented in the FreeBSD, Linux, or Solaris kernels and so on.[11] And some of different approaches for reducing the OS latency is used by systems, such as RTLinux, RTAI, and KURT and so on.

Another focus on the kernel performance has been on the overall performance optimization issues on general purpose or overall throughput which could be an issue on enterprise servers. [9][10]

The FASS(Fast Address-Space Switching for ARM Linux Kernels) is one of the a few studies on embedded systems.[3] The project aims to utilize some of the features of the Memory Management Unit in the StrongARM architecture to improve the performance of context switches under ARM Linux Kernel, although it is known to be unstable as it should be: TLB sharing does not always work. It is based on using domain tags as address-space identifiers and delaying cache flushes until a clash of mappings is detected. And they implemented TLB entries for shared pages even though the TLBs of the ARM are quite small and a potential bottleneck.

7. CONCLUSION

In this paper, we compared the context switching time and IPC performance of uClinux and Linux on the same hardware platform with ARM9 core, which is the mostly used Linux embedded system platform.

With the series of benchmark programs, uClinux showed much improved performance of context switching delay and IPC than Linux. This comes from the virtual address usage for cache architecture and the virtual address space support of Linux kernel which needs invalidation of the whole caches which makes a fixed amount of cache-miss load whenever switching the contexts of processes. uClinux which supports singular address space boosts the efficiency of cache even if context switching occurs and dramatically reduced the required delay. uClinux showed much better performance on the IPC performance also.

The uClinux will show significant benefits on a sort of applications which needs high context switching rates and significant sharing like IPC as the time critical embedded systems does.

REFERENCES

[1] Steve Furber. “ARM System-on-Chip Architecture”. Addison-Wesley, 2000.

[2] uClinux developer forum. *http://www.ucdot.org/*

[3] Adam Wiggins et el. “Implementations of Fast Address-Space Switching and TLB Sharing on the StrongARM Processor”, in the Proceedings of the 8th Australia-Pacific Computer Systems Architecture Conference, Aizu-Wakmatsu City, Japan, September 2003.

[4] HyokSung Choi. “uClinux/ARM 2.6 Project” *http://opensrc.sec.samsung.com/*

[5] McVoy, L., Staelin, C. “lmbench: Portable tools for performance analysis”. In: Proceedings of the 1996 USENIX Technical Conference, San Diego, CA, U.S. (2996)

[6] Samsung S3C24A0 Product Datasheet. *http://www.samsung.com/Products/Semiconductor/SystemLSI/MobileSolutions/MobileASSP/MobileComputing/S3C24A0/S3C24A0.htm*

[7] ARM926EJ-S Technical Reference Manual. *http://www.arm.com/pdfs/DDI0198D_926_TRM.pdf*

[8] ARM Architecture Reference Manual. ARM LTD.

[9] Duc Vianney, Sandra Johnson, Bill Hartner. “Linux Kernel Performance Measurement and Evaluation” LinuxWorld / San Francisco. Aug. 2002.

[10] Sandra K. Johnson, William H. Hartner, William C. Brantley. “Improving Linux kernel performance and scalability”. LTC. Jan 2003

[11] Luca Abeni, Ashvin Goel, Charles Krasic, Jim Snow, Jonathan Walpole. “A Measurement-Based Analysis of the Real-Time Performance of Linux”. Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium, 2002.